COP 3330: Object-Oriented Programming Summer 2007

Introduction to Object-Oriented Programming Part 2

Instructor : Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 823-2790
http://www.cs.ucf.edu/courses/cop3330/sum2007

School of Electrical Engineering and Computer Science
University of Central Florida

Another Applet Program -- **StickFigureApplet.java**

```
// A Java applet program which draws a stick figure body
import java.awt.*;
import java.applet.Applet;

public class StickFigureApplet extends Applet {

          public void paint (Graphics page) {
             page.drawString("A Stick Figure", 100,30);
             // Head
             page.drawOval(100,50,50,50);
             page.drawOval(115,65,5,5);    // eyes
             page.drawOval(130,65,5,5);
             page.drawLine(125,70,125,80); // nose
             page.drawLine(120,85,130,85); // mouth
             // Body
             page.drawLine(125,100,125,150);
             // Legs
             page.drawLine(125,150,100,200);
             page.drawLine(125,150,150,200);
             // Hands
             page.drawLine(125,125, 75,100);
             page.drawLine(125,125,175,125);
          }  // end of paint method
}   // end of class
```


Another Applet Program (cont.)

StickFigureApplet.html

```
<html>
<applet code="StickFigureApplet.class" width=300 height=300>
</applet>
</html>
```

drawString(astring,x,y)

writes the given string starting from the <x,y> coordinate.

Another Applet Program (cont.)

drawLine(x1,y1,x2,y2)

draws a line from <x1,y1> to <x2,y2> coordinate.

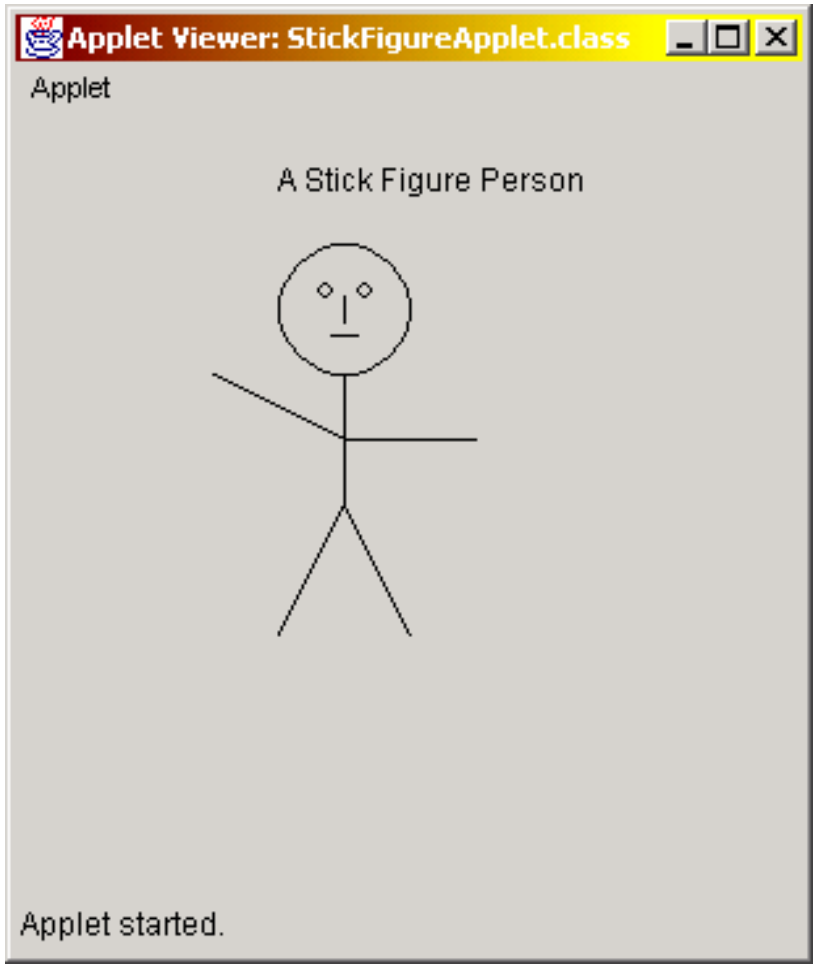
drawOval(x,y,width,height)

draws an oval with given width and height

(if the oval were enclosed in a rectangle).

<x,y> gives the top left corner of the rectangle.

Output of StickFigureApplet



JAVA API (Application Programming Interface)

- The *Java API* is a set of a class libaries.
- The classes of the Java API are grouped into packages.
- Each package contains related classes.
- A package may contain another packages too.
- We can access a class explicitly:
  `java.lang.System` (the `.` separates packages and classes).
- Or, we can access all classes in a package at the same time: `java.awt.*`

JAVA API (cont.)

- Some packages in the Java API.
  - `java.lang`
    - general support, it is automatically imported into all Java programs
  - `java.io`
    - perform a wide variety of input output functions
  - `java.awt`
    - graphics related stuff (awt-Abstract Windowing Toolkit)
  - `java.applet`
    - to create applets
  - `java.math`
    - mathematical functions.

The `import` Statement

- We can access a class by giving its full name such as `java.awt.Graphics`. But we will repeat this over and over again in our programs.

- The `import` statement identifies the packages and the classes of the Java API that will be referenced in our programs.

The `import` Statement (cont.)

`import` *`package`*`.`*`class`*

identify the particular package that will be used in our program.

example: `import java.applet.Applet`

`import` *`package`*`.*`

we will be able to access all classes in that package.

example: `import java.awt.*`

Structure of a Console Application Program

imported classes

– You should at least import classes in `java.io` package.

`public class` *<your application name>* `{`

- *declarations*

  • you should declare all variables which will be used in your methods

```
public static void main (String args[]) throws
IOException {
```

- *declarations of local variables*
- *executable statements*

```
}
```

other methods if they exist

```
}
```


Declarations

- Each declaration statement declares one or more references (references to objects) or variables (primitive data types).

Examples:

```
int x, y, z;
Graphics p1;
int A;
int b;
```


Executable Statements

- All statements which are executed during the execution of a program
  -- assignment statement
  -- method call statement
  -- if statement
  -- while statement

Examples:

```
System.out.println("Hello there, World!");
f(1,2);
x = y+2;
if (x<1) x=x+1;
    else x=x-1;
```


Structure of an Applet Program

- ***imported classes***
  - you should import at least `Graphics` and `Applet` classes

`public class <`*`your class name`*`> extends Applet {`

  - ***declarations***
    - you should declare all variables which will be used in your methods

  - ***declarations of methods in your application***
    - Declarations of your own methods and the methods responding to events.
    - If a required method is needed but it is not declared, it is inherited from Applet class. Normally the free versions we get from `Applet` class.

`}`

Some Methods for Events

- Normally, you may declare following methods (or other methods) to respond to certain events:

public void init()

– it is called when your applet is started.

-- It performs initialization of an applet.

public void paint (Graphics g)

-- it is called after the initialization.

-- it is also called automatically every time the applet needs to be repainted.

public void destroy()

-- it is called when the applet is destroyed

Structure of a Method

`public` *<its type>* *<its name>* (*<its arguments>*)

{

– *declarations of local variables*

– *executable statements*

}

Another Simple Console Application Program

```
public class Example3
{  public static void main(String args[])
   {
   // print the city and its population.
   System.out.println("The name of the city is ” + “Orlando”);
   System.out.println(“Its population is “ + 900000);

   // Different usage of + operator
   System.out.println(“Sum of 5+4: “ + (5+4));

   // Different output method – print
   System.out.print(“one..”);
   System.out.print(“two..”);
   System.out.println(“three..”);
   System.out.print(“four..”);
   } // end of main

}  // end of Example class
```


Another Simple Application Program (cont.)

- The operator + is a string concatenation operator.
  - `"abc"+"de"` ➔ `"abcde"`
  - `900000` is converted to a String ("900000") , and this string is concatenated with the string literal `"Its population is"`.

- The operator + is also a regular add operator for numeric values. The + in (5+4) is a regular addition operator for numeric values.

Another Simple Application Program (cont.)

- In other words, the + operator is overloaded.
- `println` prints its argument and moves to the next line.
- `print` prints its argument and it does not move to the next line.
- The output of our program will be:

```
The name of the city is Orlando
Its population is 900000
Sum of 5+4:  9
one..two..three..
four..
```

Simple IO

System.out.print(“A simple sentence");

The other version of this method is the println method.
The only difference between print and println is that println advances the cursor to the next line.

Thus, the following three statements:

```
System.out.print("Roses are red,");
System.out.println(" violets are blue,");
System.out.println("This poem is as good as new.");
```

would produce the following output:

```
Roses are red, violets are blue,
This poem is as good as new.
```


Simple Output

```
System.out.println("Hello " + "World!");
Output:
Hello World

System.out.println("Answer = " + 7);
Output:
Answer = 7
```

```
System.out.println(3+4);
Output:
7

System.out.println(3+4+"= 7");
Output:
7=7
```


Introduction to Objects

- Initially, we can think of an *object* as a collection of services that we can tell it to perform for us
- The services are defined by methods in a class that defines the object

```
System.out.println ("Make the most of each day.");
```

object **method** **Information provided to the method (parameters)**

The println and print Methods

- The `System.out` object provides another service as well
- The `print` method is similar to the `println` method, except that it does not advance to the next line
- Therefore anything printed after a `print` statement will appear on the same line

Variables

- A *variable* is a name for a location in memory
- A variable must be *declared*, specifying the variable's name and the type of information that will be held in it


```
int total;

int count, temp, result;
```

Multiple variables can be created in one declaration

Variables

- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```

- When a variable is referenced in a program, its current value is used

Assignment

- An assignment statement changes the value of a variable

```
total = 55;
```

- The assignment operator is the = sign
- The expression on the right is evaluated and the result is stored in the variable on the left
- The value that was in `total` is overwritten
- You can only assign a value to a variable that is consistent with the variable's declared type

Constants

- A constant is an identifier that is similar to a variable except that it holds one value for its entire existence
- The compiler will issue an error if you try to change a constant
- In Java, we use the `final` modifier to declare a constant

```
final int MIN_HEIGHT = 69;
```

- Constants:
  - give names to otherwise unclear literal values
  - facilitate changes to the code
  - prevent inadvertent errors

Constants (cont.)

- A constant is similar to a variable except that they keep the same value through their existence.
- Constants cannot be used in the left side of an assignment statement.
- They are specified using the reserved word `final` in declarations.
- Examples:

```
final double PI = 3.14159;

final int NUMOFSTUDENTS = 58;
```

Constants (cont.)

- As a style, we choose upper case letter for identifiers representing constants.
- Constants are better than literals because:
  - they make code more readable by giving a name to a value.
  - they facilitate easy updates in the programs because the value is only specified in one place.

Data Types

- Each value in memory is associated with a specific data type.
- Data type of a value determines:
  - size of the value (how many bits) and how these bits are interpreted.
  - what kind of operations we can perform on that data.
- A data type is defined by a set of values and the operators you can perform on them.
- Data Types in Java:
  - Primitive Data Types
  - Object Data Types

Primitive Data Types

- There are exactly eight primitive data types in Java
- Four of them represent integers:
  - `byte`, `short`, `int`, `long`
- Two of them represent floating point numbers:
  - `float`, `double`
- One of them represents characters:
  - `char`
- And one of them represents boolean values:
  - `boolean`

Java numerical primitive types

Type	Storage	Min Value	Max Value
byte	8 bits	-128	127
short	16 bits	-2^{15}	$2^{15}-1$
int	32 bits	-2^{31}	$2^{31}-1$
long	64 bits	-2^{63}	$2^{63}-1$
float	32 bits	-3.4e+38 (-1.4e-45)	+3.4e+38 (+1.4e-45)
double	64 bits	-1.8e+308 (-4.9e-324)	+1.79e+308 (+4.9e-324)

They differ by the amount of the memory used to store them.
Internally, integers are represented using two's complement representation (discussed later).

Literal	Assigned type
6	int (default type)
6L	long
6l	long
1,000,000,000	int
2.5 (or 2.5e-2)	double (default type)
2.5F	float
2.5f	float
‘x’	char
‘\n’	char (new line)
‘\u0039’	char represented by unicode 0039
true	boolean
“\tHello World!\n”	String (not a primitive data type!)

Assignment actually assigns value to a name

```
myNumber=4;
firstLetterOfMyName='M';
isEmpty=true;
```

Combination of declaration and assignment is called definition

```
boolean isHappy=true;
double degree=0.0;
String s = "This is a string.";
float x, y=4.1, z=2.2;
```


Binary Numbers

- Before we discuss the details of the primitive data types in Java programming language, let's review the binary numbers.

- a sequence of 0s and 1s.
- two bits: 00 01 10 11 (four different values)
- three bits: 000 001 010 011 100 101 110 111 (eight different values)
- 8 bits (1 byte): 00000000 ... 11111111 (256 different values)
- n bits: 2^n different values

Internal Data Representation of Integers

- Two's complement format is used to represent integer numbers.
- Two's complement format representation makes internal arithmetic processing easier.
- In Two's complement format:
  - positive numbers are represented as a straight forward binary number.
  - a negative value is represented by inverting all the bits in the corresponding positive number, then adding 1.

Internal Data Representation of Integers

Example: (assume 1 byte representation)

In decimal: 6 – 4 = 2

In binary: 00000110 – 00000100 = 00000010

Using Two's complement and addition we have:

- form the 2's complement of 4
  - 00000100 = 11111100
- perform addition
  - 00000110 + 11111100 = 100000010
  - overflow bit is discarded leaving: 00000010, which is the answer.

Internal Data Representation – More Examples

31: 00011111 ➔ 11100000 + 1 = 11100001 (-31)

0: 00000000

127: 01111111 ➔ 10000000 + 1 = 10000001 (-127)

1: 00000001 ➔ 11111110 + 1 = 11111111 (-1)

10000000 (which number?) ➔ -128

128:10000000 ➔ 01111111 + 1 = 10000000 (-128)

Overflow - Underflow

- If a value grows so large that it cannot be stored in the space, this is called as *OVERFLOW*.
- If a value grows so small that it cannot be stored in the space, this is called as *UNDERFLOW*.
- If an overflow or an underflow occurs, we will get incorrect results (not error messages).

Overflow:

```
byte x = 127;
x = (byte) (x+1);
```

➔ value of x is –128

01111111 + 00000001 = 10000000

Underflow:

```
byte x = -128;
x = (byte) (x-1);
```

➔ value of x is 127

10000000 + 11111111 = 01111111

Floating Point Numbers (Real Numbers)

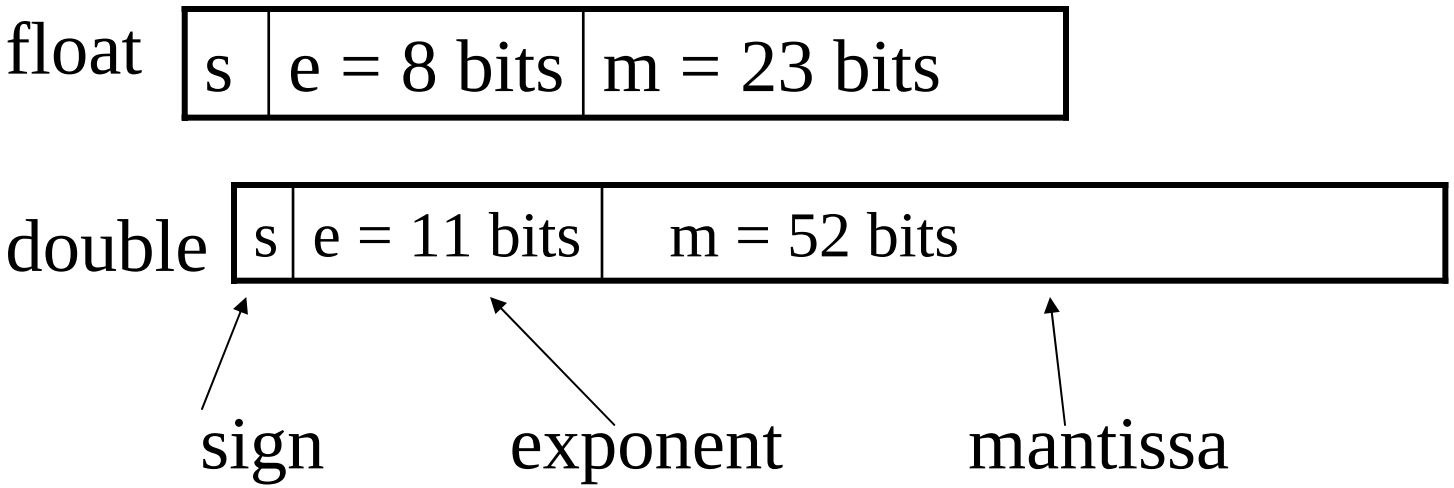
- There are two separate floating point primitive data types.
- They differ by the amount of the memory used to store them.

type	***size***	***approx. min value***	***approx. max value***	***significant digits***
float	32 bits	-3.4×10^{38}	3.4×10^{38}	7
double	64 bits	-1.7×10^{308}	1.7×10^{308}	15

- Java treats any floating point literal as double.
  - If we want to force a literal to be float: 1.2f 1.2F
  - double literals: 1.2 1.2d 1.2D

Internal Representation of Floating Point Numbers

- Java uses the standard IEEE 754 floating point format to represent real numbers.



value = sign * mantissa * $2^{exponent}$

Boolean

- A `boolean` value represents a true or false condition.

- The reserved words `true` and `false` are only valid values for a boolean type.

```
int i, j;
boolean x, y;
x = true;
y = (i < j);
```


Characters

- A `char` value stores a single character from Unicode character set.
- A character set is an ordered list of characters. Each character is represented by a sequence of bits.
- The Unicode character set uses 16 bits per character (65636 unique characters) and contains international character sets from different languages, numbers, symbols.

Characters (cont.)

- ASCII character set is a subset of the Unicode character set. It uses only 8 bits (256 characters). In fact, the first 256 characters of the Unicode character set are ASCII characters.
  - 32 space
  - 48-57 0 to 9
  - 65-90 A to Z
  - 97-122 a to z
- Character literals:
  - ‘a’ ‘A’ ‘1’ ‘0’ ‘+’
  - Note that ‘1’ and 1 are different literals (character and integer)

Reserved Words

- ***Reserved words*** are identifiers that have a special meaning in a programming language.
- For example,
  - `public, void, class, static` are reserved words in our simple programs.
- In Java, all reserved words are lower case identifiers (Of course we can use just lower case letters for our own identifiers too)
- We cannot use the reserved words as our own identifiers (i.e. we cannot use them as variables, class names, and method names).

Reserved Words (cont.)

- **Data declaration:** `boolean, float, int, char`
- **Loop keywords:** `for, while, continue`
- **Conditional keywords:** `if, else, switch`
- **Exceptional keywords:** `try, throw, catch`
- **Structure keywords:** `class, extends, implements`
- **Modifier and access keywords:** `public, private, protected`
- **Miscellaneous:** `true, null, super, this`

Sample Java Program - Example

```
/*
   HelloWorld application program
*/
public class HelloWorld                  // Class header
{                                        // Start class body
    public static void main(String argv[])      //main method
    {
            System.out.println("HelloWorld!");
     }      // end of main
}         // end HelloWorld
```

Circled words are those that we make up ourselves

Sample Java Program - Example

```
/*
   HelloWorld application program
*/
public class HelloWorld                    // Class header
{                                          // Start class body
   public static void main(String argv[])      //main method
    {
            System.out.println("HelloWorld!");
     }      // end of main
}         // end HelloWorld
```

Circled words are reserved for special purposes in the language are called **reserved words**

Sample Java Program – Example

```
/*
   HelloWorld application program
*/
public class HelloWorld                  // Class header
{                                        // Start class body
    public static void main(String argv[])      //main method
    {
            System.out.println ("HelloWorld!");
     }      // end of main
}          // end HelloWorld
```

Circled words that are not in the language, but were used by other programmers to make the library

Class Libraries

- A *class library* is a collection of classes that we can use when developing programs
- There is a *Java standard class library* that is part of any Java development environment
- These classes are not part of the Java language per se, but we rely on them heavily
- The `System` class and the `String` class are part of the Java standard class library
- Other class libraries can be obtained through third party vendors, or you can create them yourself

Packages

- The classes of the Java standard class library are organized into packages
- Some of the packages in the standard class library are:

Package	Purpose
`java.lang`	**General support**
`java.applet`	**Creating applets for the web**
`java.awt`	**Graphics and graphical user interfaces**
`javax.swing`	**Additional graphics capabilities and components**
`java.net`	**Network communication**
`java.util`	**Utilities**

The `import` Declaration Revisited

- All classes of the `java.lang` package are automatically imported into all programs
- That's why we didn't have to explicitly import the `System` or `String` classes in earlier programs
- The `Random` class is part of the `java.util` package
- It provides methods that generate pseudo-random numbers
- We often have to *scale* and *shift* a number into an appropriate range for a particular purpose

Wrapper Classes

- For each primitive data type, there exists a *wrapper class*.
- A wrapper class contains the same type of data as its corresponding primitive data type, but it represents the information as an object (an instance of that wrapper class).
- A wrapper class is useful when we need an object instead of a primitive data type.
- Wrapper classes contain useful methods. For example, `Integer` wrapper class contains a method to convert a string which contains a number into its corresponding value.

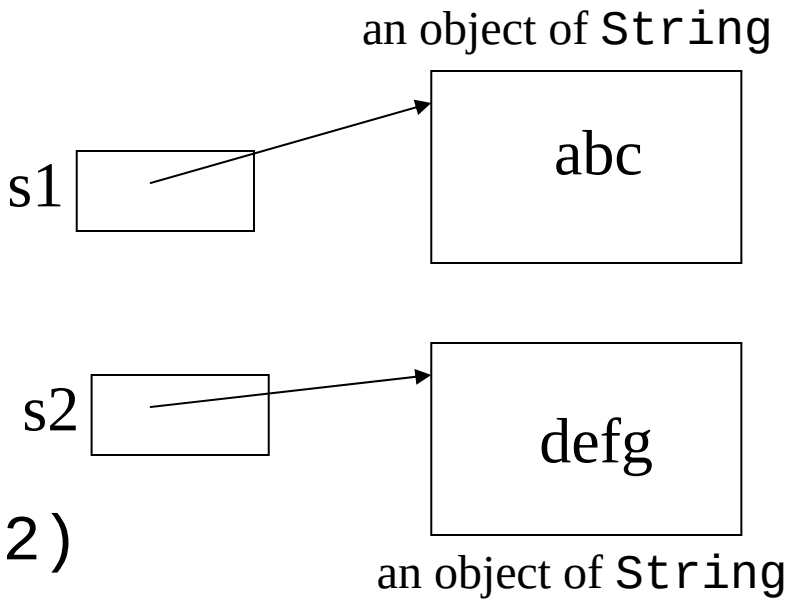
Wrapper Classes (cont.)

- When we talk numeric input/output, we will use these wrapper classes.
- Wrapper Classes:
  - `Byte Short Integer Long Float Double Character Boolean Void`

An Object Data Type (String)

- Each object value is an instance of a class.
- The internal representation of an object can be more complex.
- We will look at the object data types in detail later.
- String literals: “my name” “123”

```
String s1, s2;
s1 = “abc”
s2 = “defg”;

System.out.println(s1+s2)
```



The String Class

- Every character string is an object in Java, defined by the `String` class
- Every string literal, delimited by double quotation marks, represents a `String` object
- The *string concatenation operator* (+) is used to append one string to the end of another
- It can also be used to append a number to a string
- A string literal cannot be broken across two lines in a program

The String Class in Java

String(String str); //constructor

char charAt(int index);

int compareTo(String str);

String concat(String str);

boolean equals(String str);

boolean equalsIgnoreCase(String str);

int length();

String replace(char oldChar, char newChar);

String substring(int offset, int endIndex);

String toLowerCase();

String toUpperCase();

String Concatenation

- The plus operator (+) is also used for arithmetic addition
- The function that the + operator performs depends on the type of the information on which it operates
- If both operands are strings, or if one is a string and one is a number, it performs string concatenation
- If both operands are numeric, it adds them
- The + operator is evaluated left to right
- Parentheses can be used to force the operation order

Escape Sequences

- What if we wanted to print a double quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println ("I said "Hello" to you.");
```

- An *escape sequence* is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\), which indicates that the character(s) that follow should be treated in a special way

```
System.out.println ("I said \"Hello\" to you.");
```

Java Escape Sequences

- Some Java escape sequences.

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

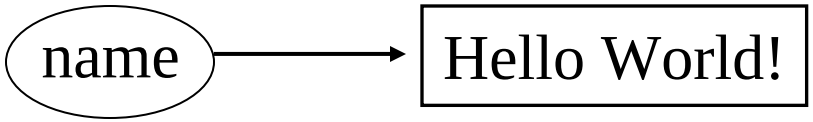
The String Type in Java

- The type for arbitrary text
- Is not a primitive data type, but Java has String literals
- Strings are objects, represented by String class in java.lang package

declaration

instantiation

- String name;
  name=new String (" Hello World!");



- String name=new String("Hello World!");

constructor

```
public class StringClass
{
    public static void main(String [] args)
    {
        String phrase=new String("This is a class");
        String string1, string2, string3, string4;
        char letter;
        int length=phrase.length();
        letter = phrase.charAt(5);
        string1=phrase.concat(", which manipulates strings");
        string2=string1.toUpperCase();
        string3=string2.replace('E', 'X');
        string4=string3.substring(3, 30);
        System.out.println("Original string:"+phrase);
        System.out.println(letter);
        System.out.println("length is"+length);
     ....
    }
}
```


Arithmetic Expressions

- Simple Assignment Statements:

```
x = y + z;
x = x * 5;
```

- Some of Arithmetic Operators:
  - `+` addition
  - `-` subtraction
  - `*` multiplication
  - `/` division
  - `%` mod operator (remainder)

Arithmetic operators (cont.)

`op1+op2`	**addition**
`op1-op2`	**subtraction**
`op1*op2`	**multiplication**
`op1/op2`	**division**
`op1%op2`	**modulo**

- op1 and op2 can be of integer or floating-point data types
- if op1 and op2 are of the same type, the type of result will be the same
- mixed data types ⟹ arithmetic promotion before evaluation
- op1 or op2 is a string ⟹ **+** operator performs **concatenation**

Division

- If the operands of the / operator are both integers, the result is an integer (the fractional part is truncated). If one or more operands of the / operator are floating point numbers, the result is a floating point number.
- The remainder operator % returns the integer remainder after dividing the first operand by the second one. The operands of % must be integers.
- Examples:

13 / 5	➔	2
13.0 / 5	➔	2.4
13 / 5.0	➔	2.4
2 / 4	➔	0
2.0 / 4.0	➔	0.5
6 % 2	➔	0
14%5	➔	4
-14%5	➔	-4

Operator Precedence

```
x = x + y * 5;                    // what is the order of
    evaluation?
```

- Operators in the expressions are evaluated according to the rules of precedence and association.
  - Operators with higher order precedence are evaluated first
  - If two operators have same precedence, they are evaluated according to association rules.
  - Parentheses can change the order of the evaluations.

Operator Precedence (cont.)

- Precedence Rules for some arithmetic operators:

`+ -` (unary minus and plus)	right to left	higher
`* / %`	left to right	
`+ -`	left to right	lower

- Examples:

```
x = a + b * c – d;      ➔    x = ((a+(b*c))-d);

x = (a + b) * c – d;    ➔    x = (((a+b)*c)-d);

x = a + b + c;          ➔    x = ((a+b)+c);
```


Data Conversion

- Because Java is a strongly typed language, each data value is associated with a particular type.
- Sometimes we may need to convert a data value of one type to another type.
- In this conversion process, we may use loose important information.
- A conversion between two primitive data types falls into one of two categories:
  - *widening conversion* – widening conversions usually do not loose information
  - *narrowing conversion* – narrowing conversions may loose information
- A boolean value cannot be converted to any other primitive type.

Java Widening Conversions

- In widening conversions, they often go from one type to another type that uses more space to store the value.
- In the most of widening conversions, we do not loose information.
  - we may loose information in the following widening conversions:
    - int ➔ float long ➔ float long ➔ double

From	To
byte	short, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	float, double
float	double

Widening Conversions (cont.)

- A widening conversion may automatically occur:

```
int x; long y; double z;
y = x;
z = x + 1;        // the result of the addition is converted into double
Z = x + 1.0;  // the value of x is converted to double, then the addition is performed.
```

- We may loose information in some widening conversions;

```
int x = 1234567891;    // int is 32-bit and float is 32-bit
float y;
y = x;     // we will loose some precision (7 digit precision)
```


Narrowing Conversions

- In narrowing conversions, we may loose information.
- In narrowing conversions, they often go from one type to another type uses less space to store the value.
- We treat the conversion from `byte` to `char` as a narrowing conversion, because we loose the negative sign of a byte value.

From	To
`byte`	`char`
`short`	`byte, char`
`char`	`byte, short`
`int`	`byte, short, char`
`long`	`byte, short, char, int`
`float`	`byte, short, char, int, long`
`double`	`byte, short, char, int, long, float`

Type Conversion

- In Java, type conversions can occur in three ways:
  - **assignment conversion** – the value of the expression can be automatically converted into the type of the variable (if it is a widening conversion).
  - **arithmetic promotion** – the value of an operand of an operator in an arithmetic expression can be automatically converted into another type to perform that operator (if it is a widening conversion).
  - **casting** – we explicitly convert a type into another type using type casting operation.

- Examples:

```
int x; double y; long z;
y = x;          // assignment conversion
y = x + 2.0;    // arithmetic promotion
x = (int) z;    // casting
```


Type Conversion – More Examples

```
byte x=1;
x = x + 1;  // type error – correct: x=(byte)(x+1);

int y;  double z;
z = y + 1 + z;

int x=3;
double y;
y = x / 2;
y = x / 2.0;
y = (double) x / 2;
y = (double) (x/2);
```


Increment and Decrement Operators

- The increment operator **++** and the decrement operator **--** can be applied to all integer and floating point types.
- The increment and decrement operators are unary operators, and their arguments must be variables.
- The increment operator adds one to its argument, the decrement operator subtracts one from its argument.
- They can be prefix or postfix operators.
  - pre-increment, post-increment, pre-decrement, post-decrement
  - In pre-increment and pre-decrement operations, first these operations are performed then the value of the variable is used in the expression.
  - In post-increment and post-decrement operations, first the value of the variable is used in the expression, then these operations are performed.

Increment and Decrement Operators (cont.)

- Examples

```
int x=2;   int y;
y = ++x * 2; ➔ x is 3, y is 6
y = x++ * 3; ➔ x is 4, y is 9
y = x-- + 1; ➔ x is 3, y is 5
y = --x + 1; ➔ x is 2, y is 3
```


Operators and Precedence Rules

Precedence Level	Operator	Operation	Associates
1	++ --	postfix increment, postfix decrement	L to R
2	++ -- + - ~ !	pre-increment, post-increment, unary minus and plus bitwise complement logical not	R to L
3	* / %	multiplication, division, remainder	L to R
4	+ -	addition, subtraction	L to R
5	<< >> >>>	left shift, right shift with sign, left shift with zero	L to R
6	< <= > >=	less than, less than equal, greater than, greater than equal	L to R
7	== !=	equal, not equal	L to R
8	&	bitwise and	L to R
9	^	xor	L to R
10	|	bitwise or	L to R
11	&&	logical and	L to R
12	||	logical or	L to R
13	?:	conditional operator	R to L
14	= +=	assignment operators	R to L

Input and Output (in Console Applications)

- Java I/O is based on input and output streams
- There are pre-defined standard streams
  - `System.in` reading input keyboard (InputStream object)
  - `System.out` writing output monitor (PrintStream object)
  - `System.err` writing output (for errors) monitor (PrintStream object)
- `print` and `println` methods (defined in PrintStream class) are used to write to the the standard output stream (System.out).
- We will get the inputs from the standard input stream (System.in).
- To read character strings, we will create a more useful object of BufferedReader class from System.in.

```
BufferedReader stdin =
   new BufferedReader(new InputStreamReader(System.in));
```


Simple I/O Program

```
// Developer:  Mark Llewellyn        Date: May 22, 2007
// This program will ask your name and print you a welcome message
import java.io.*;
public class WelcomeMessage
{   public static void main(String args[]) throws IOException
    {
          String yourName;

          // Create  a BufferedReader object
          BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));

          // print the prompt
          System.out.print("Enter your name and push enter key > ");
          System.out.flush();

          // read the name
                    yourName = stdin.readLine();

          // print name together with Hi
          System.out.println("Hi " + yourName + ". . . Welcome to Java!!");
    }
}
```


Numeric Input

- We always read a string first, then we convert that string into a numeric value.

```
String astring;
int num;
astring = stdin.readLine();
num = Integer.parseInt(astring);
```

- `parseInt` is a static method in the wrapper class `Integer` which converts a string into an `int` value (assuming that the string holds the digits of that integer).
- If we put space before or after the integer number, the Java system will give an error message.

Numeric Input – trim()

- To able to put extra spaces before and after a numeric value during the input, we may use `trim` method of String class.

```
num = Integer.parseInt(astring.trim());
```

- `trim` method removes blanks in the front and the end of a string object (creates a new String object).

```
String s;
s = “    123    “;
s.trim() ➔ “123”
```


To Read a double Value

- We always read a string first, then we convert that string into a numeric value.

```
String astring;
double num;
astring = stdin.readLine();
num = Double.parseDouble(astring);
```

- `parseDouble` is a static method in the wrapper class `Double` which converts a string into an `double` value (assuming that the string holds the digits of that double number).

- To able to put extra spaces before and after a numeric value during the input, we may use `trim` method of `String` class.

```
num = Double.parseDouble(astring.trim());
```

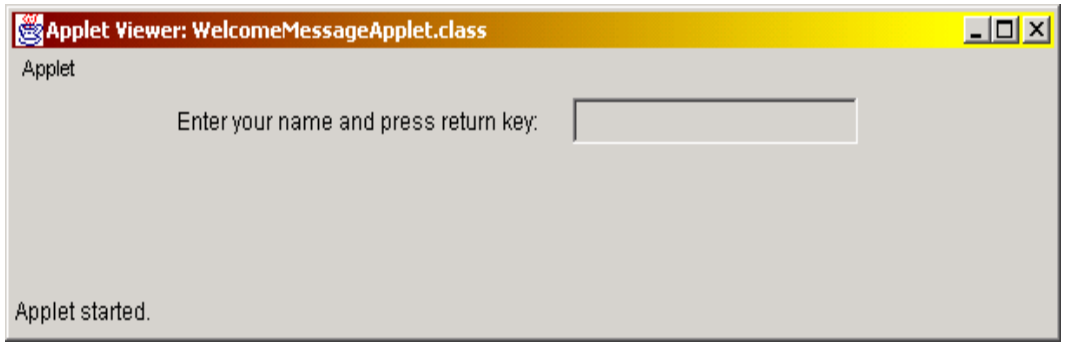

Simple I/O in Applets

```
// Developer:  Mark Llewellyn     Date: May 22, 2007
// Applet version of the Welcome application on page 77

import java.applet.*;
import java.swing.*;
import java.awt.*;
import java.awt.event.*;

public class WelcomeMessageApplet extends Applet implements ActionListener
{  Label prompt,greeting;   // Output areas
   TextField input;         // Input area
   // This method will be called when the applet starts,
   // and creates the required parts of the applet.
   public void init() {
      // Create prompt area and put it into the applet
      prompt = new Label("Enter your name and push return key: ");
      add(prompt);
      // Create input area and put it into the applet
      input = new TextField(20);
      add(input);
      // Create greeting area and put it into the applet
      greeting = new Label("");
      add(greeting);
```

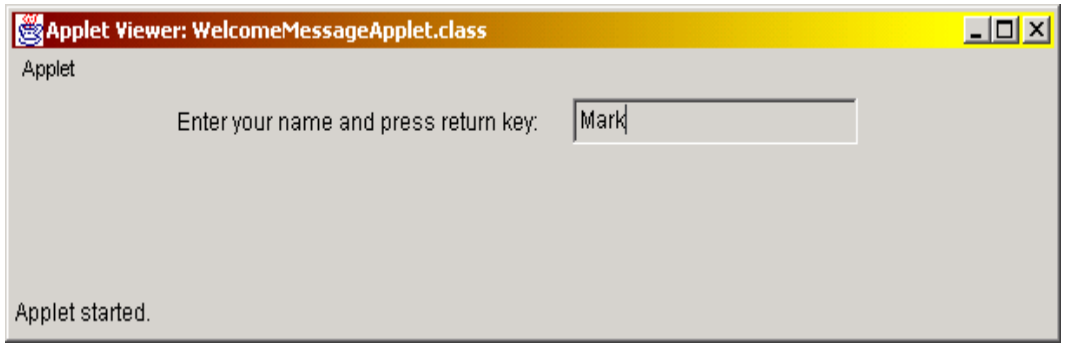

Simple I/O in Applets (cont.)

```
  // "this" (this applet) will listen the input area and
      // respond the events in this area.
      input.addActionListener(this);
}
 // This method will be called when anytime a string is typed
 // in input area and the return key is pushed.
 public void actionPerformed(ActionEvent e) {
   greeting.setSize(300,20);
   greeting.setText("Hi " + input.getText() + " Welcome to Java!!);
   input.setText("");
 }
}
```

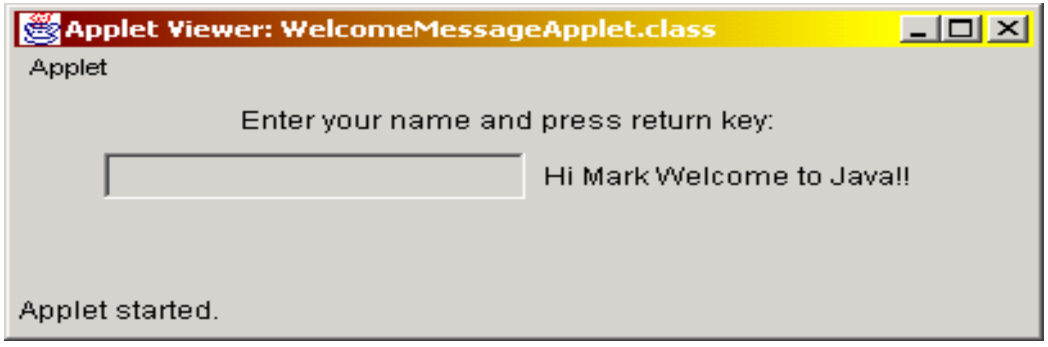

Simple I/O in Applets (Output)



At the beginning

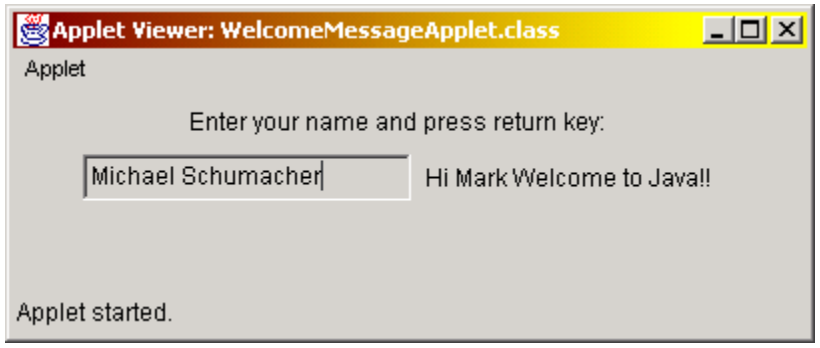


Before we push the return key

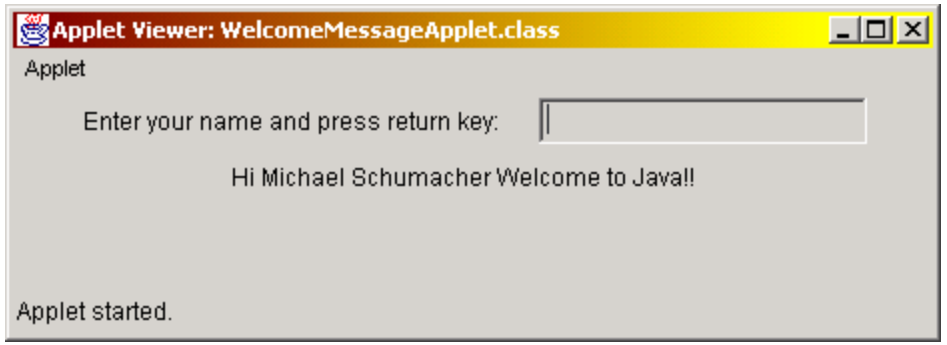


After we pressed the return key

Simple I/O in Applets (Output-cont.)



Before we push the return key



After we pushed the return key

Another Applet with I/O Operations

```
// Developer:  Mark Llewellyn          Date: May 22, 2007
// I/O demonstration using an applet

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class SumAverageApplet extends Applet implements
   ActionListener
{
   Label prompt;       // prompt area
   TextField input;    // input area for positive integer
   int number;         // value of positive integer
   int sum;            // sum of all numbers
   int count;          // number of positive integers
   double average;     // average of all numbers
```

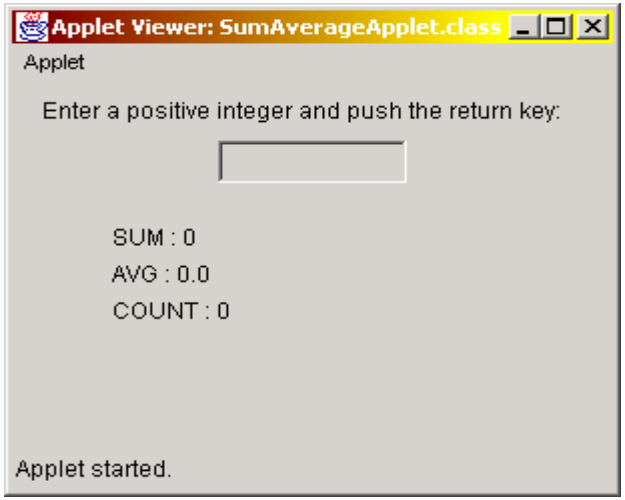

Another Applet with I/O Operations (cont.)

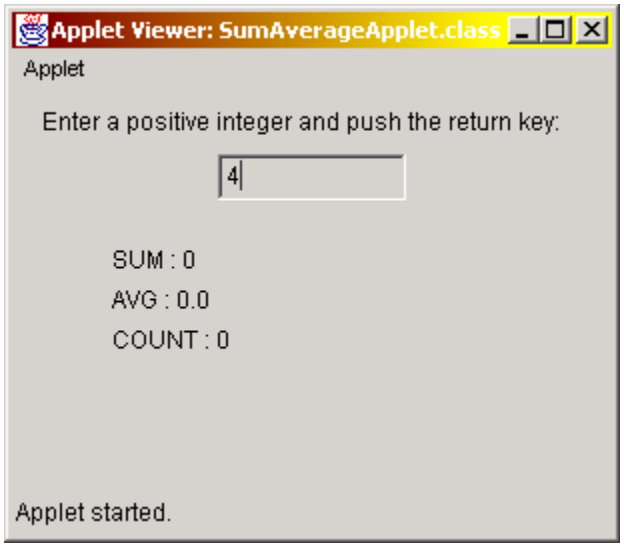
```
// Create the graphical components and initialize the variables
public void init()
   {
      // Create prompt and put it into the applet
      prompt = new Label( "Enter a positive integer and push the return key:"
   );
      add(prompt);
      // Create input area and put it into the applet
      input = new TextField(10);
      add(input);
      // Initialize the variables
      sum = 0;
      count = 0;
      // this applet will listen input area and respond to the events
    happening in this area
      input.addActionListener(this);
  }
```

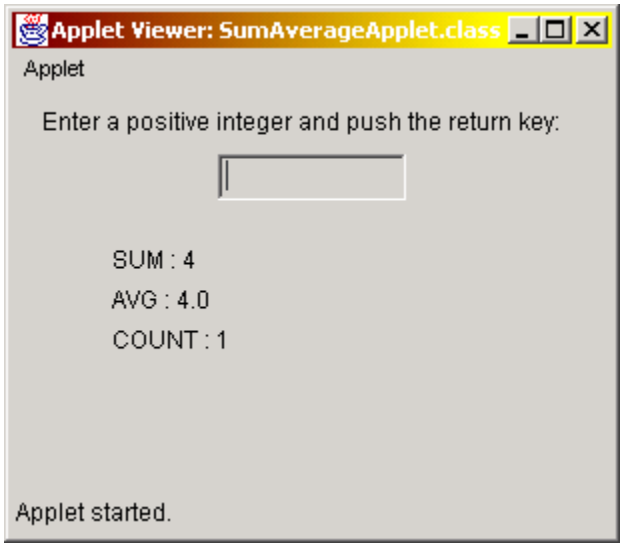

Another Applet with I/O Operations (cont.)

```
// Respond to the events in input area
public void actionPerformed(ActionEvent e)
{   // Get the input and convert it into a number
    number = Integer.parseInt(e.getActionCommand().trim());
    // Evaluate the new values
    sum = sum + number;
    count = count + 1;
   average = (double) sum / count;
    input.setText("");

    // show the results
    repaint();
}
// Show the results
public void paint(Graphics g) {
   g.drawString("SUM : " + Integer.toString(sum), 50, 50);
   g.drawString("AVG : " + Double.toString(average), 50, 70);
   g.drawString("COUNT : " + Integer.toString(count), 50, 90);
}
}
```


Another Applet with I/O Operations (output)

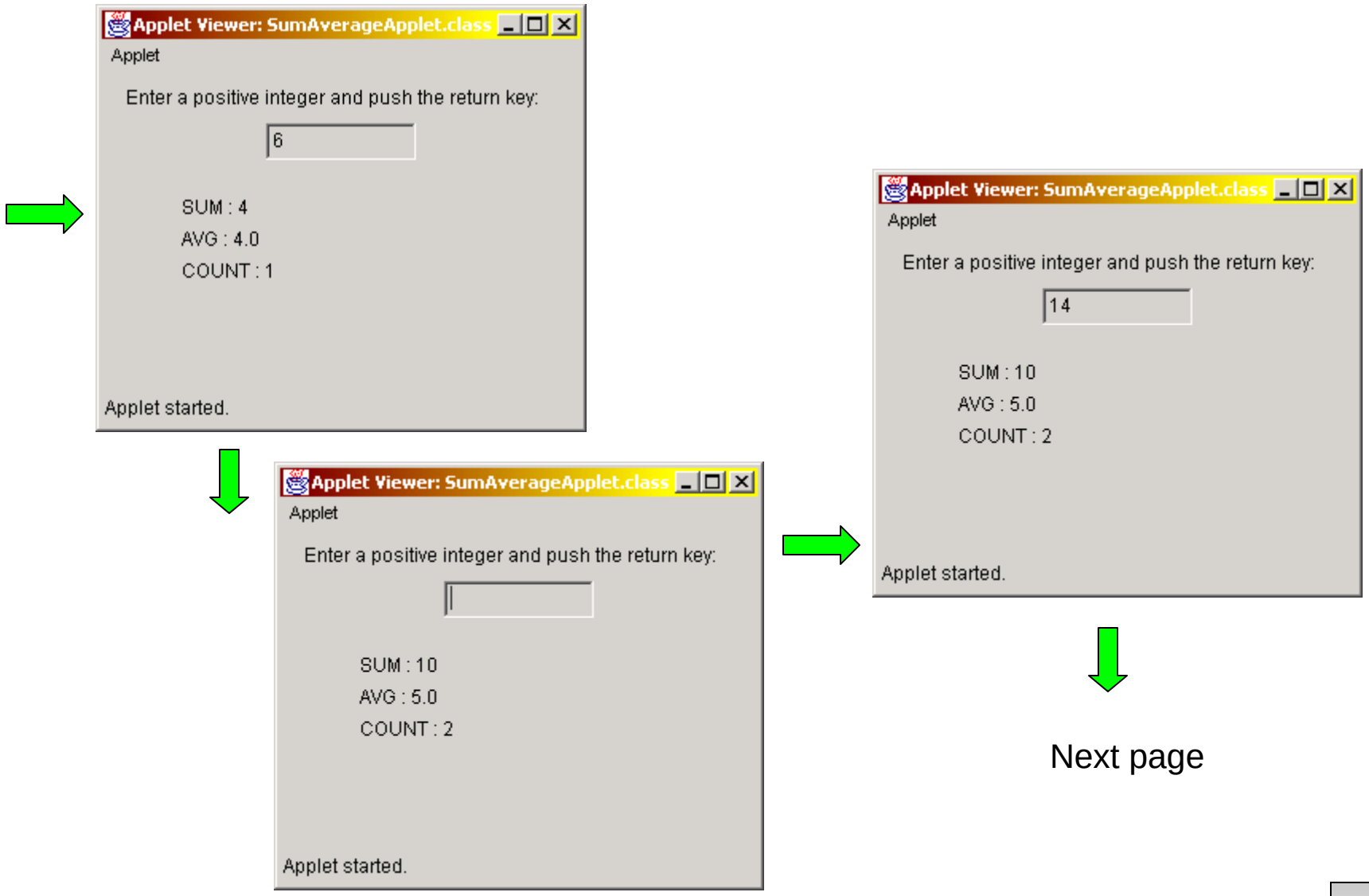






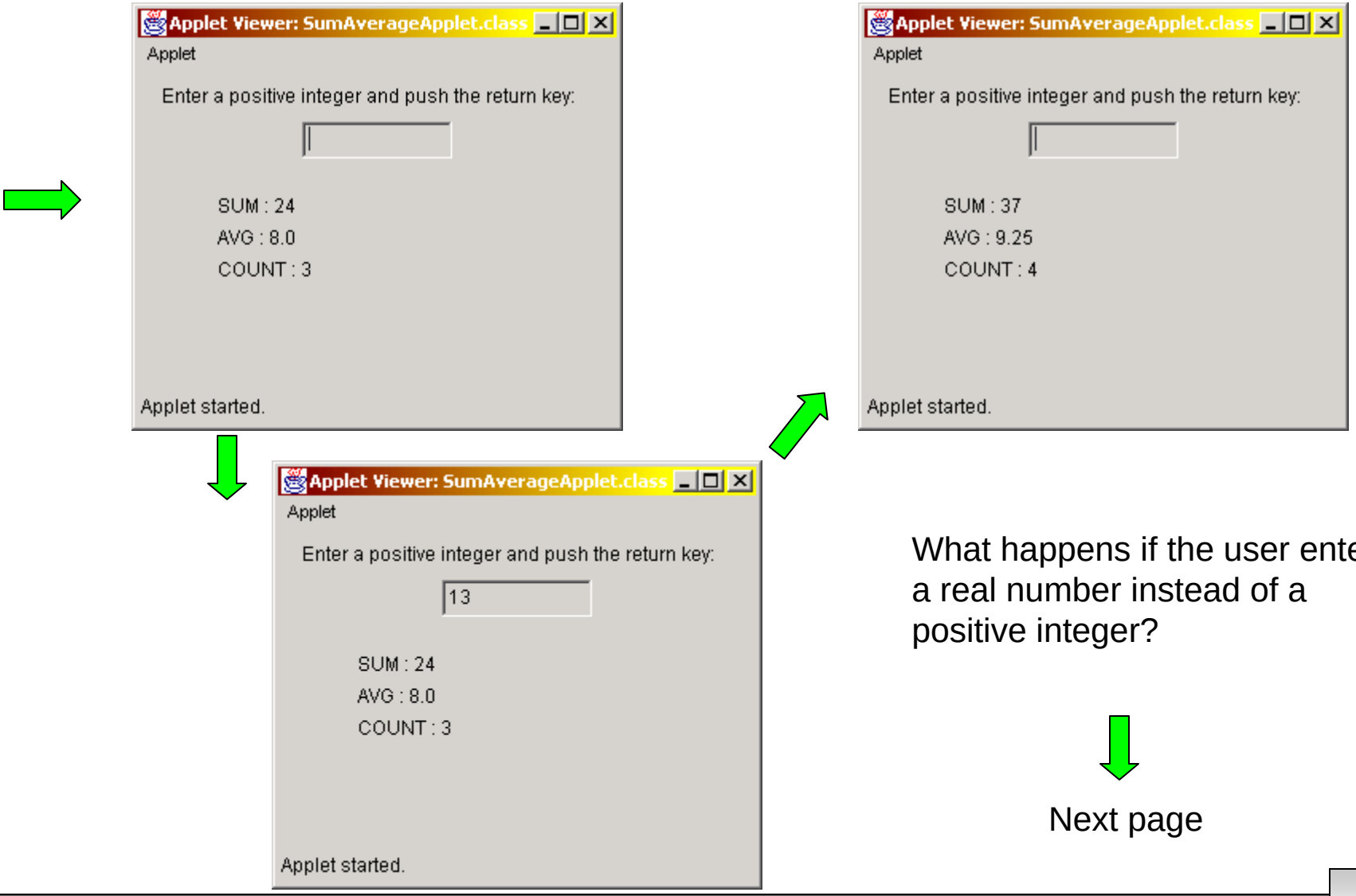
Next page

Another Applet with I/O Operations (output cont.)



Next page

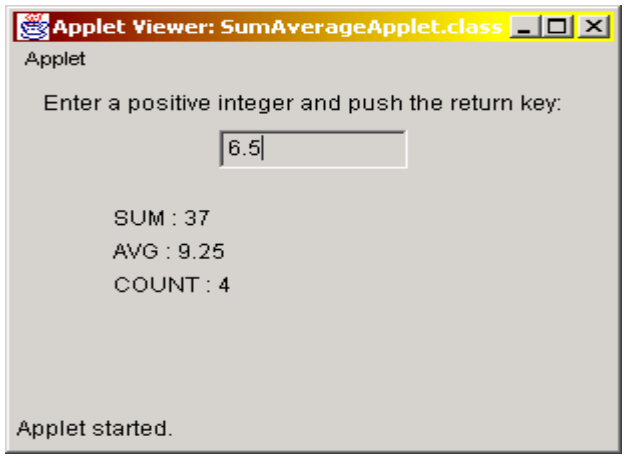
Another Applet with I/O Operations (output cont.)

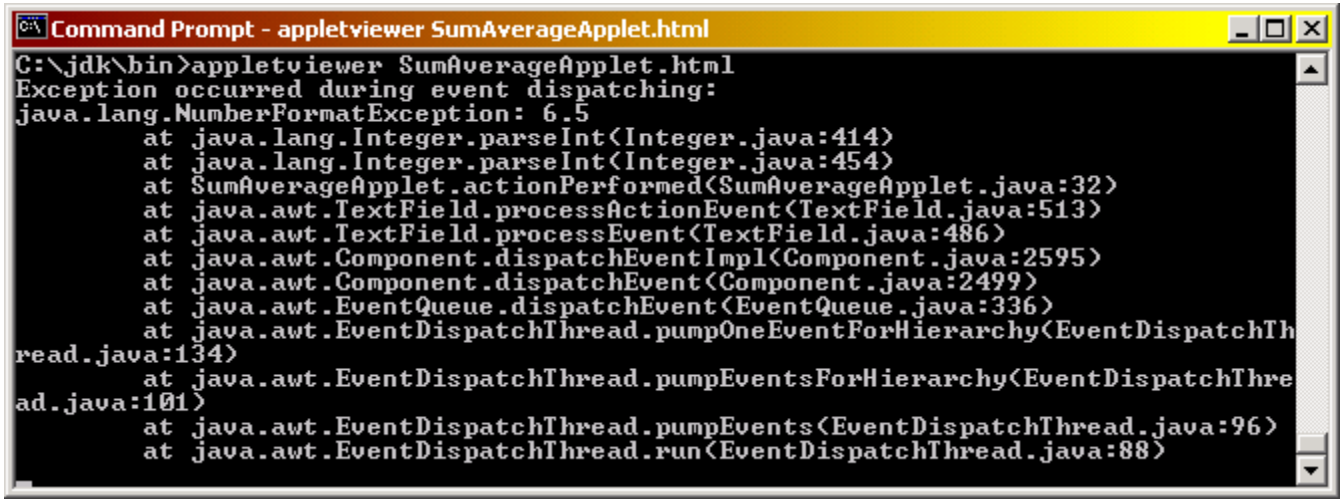


What happens if the user enters a real number instead of a positive integer?

Next page

What happens when we type a real number?





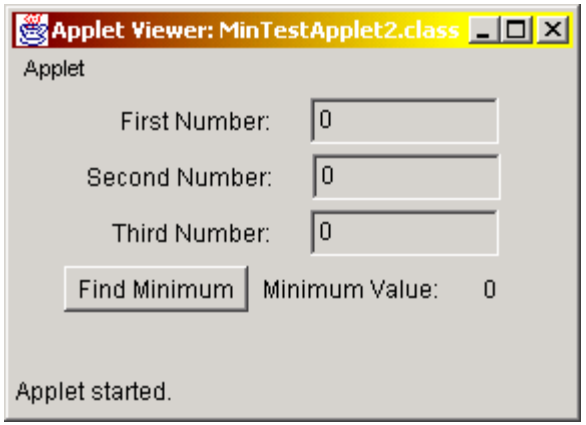
Another Applet

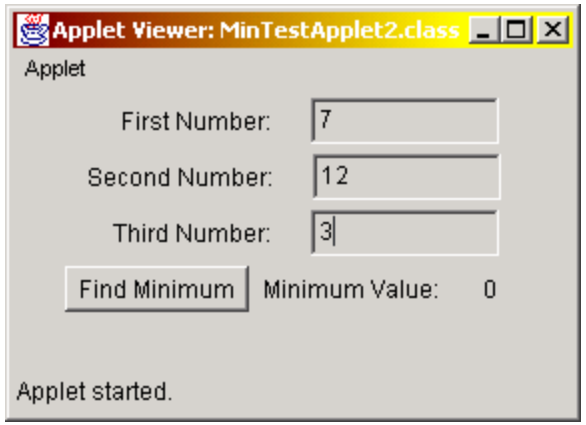
```
//Developer:  Mark Llewellyn            Date: May 22, 2007
// Finding the minimum of three integers   -- Applet Version
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class MinTestApplet extends Applet implements ActionListener {
  TextField tNum1,tNum2,tNum3;          // text fields for indegers
  Button b;
  Label lMinValue;            // Area for the minimum number
  int minValue,num1,num2,num3;
  // Create fields and initialize the variables
  public void init() {
     // Initialize variables
     minValue = 0; num1 = 0; num2 = 0; num3 = 0;
     // Create input areas
    b = new Button("Find Minimum"); add(b);
    add(new Label("First Number: "));  tNum1=new TextField("0",10); add(tNum1);
     add(new Label("Second Number: ")); tNum2=new TextField("0",10); add(tNum2);
     add(new Label("Third Number: "));  tNum3=new TextField("0",10); add(tNum3);
     // create output area
     add(new Label("Minimum Value: ")); lMinValue=new Label("0"); add(lMinValue);
     // Listen the button
     b.addActionListener(this);
}
```


Modified Applet Version (cont.)

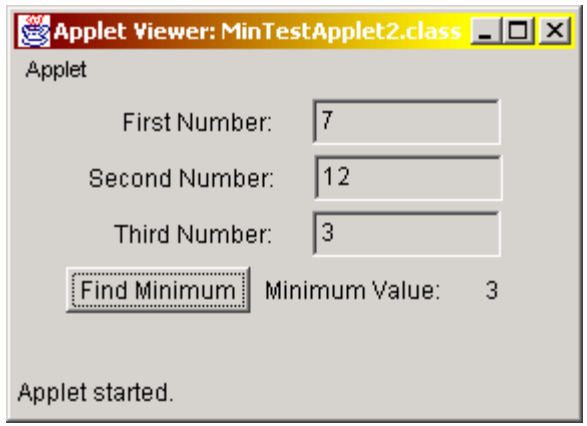
```
// Respond to the events
   public void actionPerformed(ActionEvent e) {
         // Find the source of the event
         num1 = Integer.parseInt(tNum1.getText().trim());
         num2 = Integer.parseInt(tNum2.getText().trim());
         num3 = Integer.parseInt(tNum3.getText().trim());

         // find the minumum
         if (num1<num2)
           minValue = num1;
         else
           minValue = num2;

         if (num3<minValue)
           minValue = num3;

         // put the new minimum into the applet
         lMinValue.setText(minValue + "");
   } }
```


Modified Applet Version (output)





Before pushing button



After pushing button